Computing the Maximum Detour and Spanning Ratio of Planar Paths, Trees and Cycles*

Stefan Langerman[1], Pat Morin[1], and Michael Soss[2]

[1] School of Computer Science, McGill University
3480 University St., Suite 318
Montréal, Québec, CANADA, H3A 2A7
{sl,morin}@cgm.cs.mcgill.ca
[2] Chemical Computing Group, Inc.
1010 Sherbrooke Street West, Suite 910
Montréal, Québec, CANADA, H3A 2R7
soss@chemcomp.com

Abstract. The maximum detour and spanning ratio of an embedded graph G are values that measure how well G approximates Euclidean space and the complete Euclidean graph, respectively. In this paper we describe $O(n \log n)$ time algorithms for computing the maximum detour and spanning ratio of a planar polygonal path. These algorithms solve open problems posed in at least two previous works [5,10]. We also generalize these algorithms to obtain $O(n \log^2 n)$ time algorithms for computing the maximum detour and spanning ratio of planar trees and cycles.

1 Introduction

Let $G = (V, E)$ be an embedded connected graph with n vertices and m edges. Specifically, the vertex set V consists of points in $\mathbb{R}^2$, and E consists of closed line segments whose endpoints are in V. Let s and t be two points in $\cup E$.[1] We denote by $\|st\|$ the Euclidean distance between s and t and by $\|st\|_G$ the length of the shortest path from s to t in G. The detour between two points $s, t \in \cup E$ is defined as

$$D(G, s, t) = \frac{\|st\|_G}{\|st\|} \ .$$

The *maximum detour* $D(G)$ of G is the maximum detour over all pairs of points in G, i.e.,

$$D(G) = \max\{D(G, s, t) : s, t \in \cup E, s \neq t\} \ .$$

The *spanning ratio* or *stretch factor* $S(G)$ of G is the maximum detour over all pairs of vertices of G, i.e.,

$$S(G) = \max\{D(G, s, t) : s, t \in V, \ s \neq t\} \ .$$

* This research was partly funded by CRM, FCAR, MITACS, and NSERC. This research was done while the third author was affiliated with SOCS, McGill University.

[1] Here and thoughout, $\cup S$ is shorthand for $\bigcup_{e \in S} e$.

The maximum detour and spanning ratio play important roles in the analysis of online routing algorithms [3,8] and the construction of spanners [6]. In the former case, the goal is to find paths that minimize maximum detour. In the latter, the goal is to construct graphs with few edges that minimize the spanning ratio.

1.1 Related Work

Recently, researchers have become interested in computing the maximum detour and spanning ratio of embedded graphs. The spanning ratio can be computed in $O(n(m + n \log n))$ time by computing the shortest paths between all pairs of vertices and then comparing these to the distances between all pairs of vertices. In $\mathbb{R}^2$, the maximum detour is infinite if G is non planar, so the maximum detour can be computed in $O(n^2 \log n)$ time by computing shortest paths and using this information to find the maximum detour between each pair of edges. Surprisingly, these are the best known results for computing the maximum detour or spanning ratio. Even if the input graph G is a path, no sub-quadratic time algorithms are known, though fast approximation algorithms have been reported.

Narasimhan and Smid [10] study the problem of approximating the spanning ratio in a graph and give $O(n \log n)$ time algorithms that can $(1-\epsilon)$-approximate the spanning ratio when G is a path, a cycle or a tree. More generally, they show that, after $O(n \log n)$ preprocessing, the problem of approximating the spanning ratio can be reduced to $O(n)$ approximate shortest path queries on G. Their results hold even for graphs embedded in $\mathbb{R}^d$. The authors also show that approximating the spanning ratio requires $\Omega(n \log n)$ time in the algebraic decision tree model of computation.

Ebbers-Baumann *et al* [5] study the problem of approximating the maximum detour of a polygonal path and give an $O(\frac{n}{\epsilon} \log n)$ time algorithm that finds a $(1 - \epsilon)$-approximation to the maximum detour.

1.2 New Results

In this paper we give randomized algorithms with $O(n \log n)$ expected running time that compute the exact spanning ratio or maximum detour of a polygonal path with n vertices. These are the first sub-quadratic time algorithms for finding the exact spanning ratio or maximum detour, and they solve open problems posed in at least two papers [5,10].[2]

We solve these problems by reducing the associated decision problem to computing the upper envelope of a set of identical cones in $\mathbb{R}^3$. In the case of spanning ratio, the set of cones is finite, and the upper envelope that we compute is actually an additively-weighted Voronoi diagram of points in the plane. In the case of maximum detour, the set of cones is infinite, and corresponds to computing the additively-weighted Voronoi diagram of line segments in the plane, a diagram

[2] Subquadratic time algorithms were independently found by Agarwal *et al* [1], see below.

that seems not to have been considered previously. We then apply a general optimization technique of Chan [4] to convert the decision algorithm into an optimization algorithm.

We also show that more complicated structures can sometimes be treated by using multiple invocations of the above technique. As examples, we give $O(n \log^2 n)$ time algorithms for computing the maximum detour and spanning ratio of a planar tree and $O(n \log^2 n)$ time algorithms for computing the maximum detour and spanning ratio of a planar cycle.

Independently, Agarwal *et al* [1] studied the problem of computing the maximum detour of paths, cycles and trees in two and three dimensions. For planar paths and cycles they give $O(n \log^4 n)$ time deterministic algorithms and $O(n \log^3 n)$ time randomized algorithms. For planar trees they give an $O(n \log^5 n)$ time deterministic algorithm and an $O(n \log^4 n)$ time randomized algorithm. For three-dimensional paths they give a randomized algorithm whose expected running time is $O(n^{16/9+\epsilon})$, where ϵ is any constant greater than zero.

The remainder of the paper is organized as follows: In Section 2 we show how to reduce the decision problems to an upper envelope computation. In Section 3 we give an algorithm for computing the upper envelope of a set of objects called bats that is required for solving the maximum detour decision problem. In Section 4 we describe how to use these decision algorithms to obtain optimization algorithms. In Section 5 we extend these algorithms to trees and cycles.

2 A Problem on Cones

For a point $p \in \mathbb{R}^3$, denote by p_z the z-coordinate of p and by p_{xy}, the projection of p onto the xy plane. Given a polygonal path C in $\mathbb{R}^2$ whose vertices are $v_1, \ldots, v_n$ we lift it to a polygonal path C' in $\mathbb{R}^3$ by assigning to each point p in C a z-coordinate equal to its distance along the path from v_1, i.e., for each point $p \in C$, C' has a point p' such that $p'_{xy} = p$ and $p'_z = \|v_1 p\|_C$.

In this way, we obtain a z-monotone polygonal path C' with vertices $u_1, \ldots, u_n$ such that for any two points $p', q' \in C'$, the unique path between the corresponding points $p, q \in C$ has length $\|pq\|_C = \|p'_z - q'_z\|$.

Consider the following construction. At each vertex u_i of C' we place the apex of a cone c_i that points downwards with its axis of rotation parallel to the z-axis and that spans an angle of $2 \arctan(1/d)$. Now, if some cone c_i contains some vertex u_j then $D(C, v_i, v_j) \geq d$ (see Fig. 1). Conversely, if there exists a pair of vertices v_i, v_j such that $D(C, v_i, v_j) \geq d$, then either c_i contains u_j or c_j contains u_i. Thus, the problem of determining whether the spanning ratio of C is greater than or equal to d is reducible to the problem of determining whether any cone c_i contains any vertex u_j.

The *upper envelope* of the cones $c_1, \ldots, c_n$ is the bivariate function $f(x, y) = \max\{z : (x, y, z) \in c_i, \text{ for some } i\}$. From this definition it follows that u_i is contained in some cone if and only if u_i does not appear on the upper envelope, i.e., $f(u_i) \neq u_{i,z}$. The upper envelope of identical and identically oriented cones

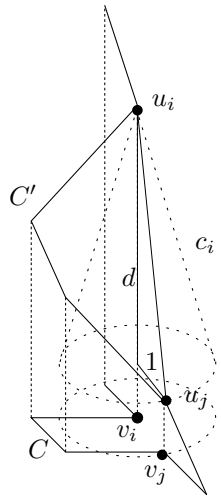

Fig. 1. If c_i contains u_j then $D(C, v_i, v_j) \geq d$.

has been given at least two other names: additively-weighted Voronoi diagram [7] and Johnson-Mehl crystal growth diagram [9]. It is known that f consists of $O(n)$ pieces and can be computed in $O(n \log n)$ time using a sweep line algorithm [7]. Thus, the decision problem of determining whether the spanning ratio of C is at least d can be solved in $O(n \log n)$ time.

Next we turn to the problem of determining whether the maximum detour of G is at least d. For this problem we use the same construction except that we place a cone with its apex on *every* point of C', not just the vertices. The decision problem then reduces to the question: Does every point on C' appear on the upper envelope of these (infinitely many) cones. Of course, computationally, we do not compute the upper envelope of infinitely many cones. Instead, we compute the upper envelope of n *bats*, where a *bat* is the convex hull of two cones with apexes on the endpoints of an edge of C. We call the edge that defines a bat the *core* of the bat.

Thus, for both maximum detour and spanning ratio, the associated decision problem can be solved by computing the upper envelope of a suitably chosen set of objects, either cones or bats. To the best of our knowledge, no algorithm exists for computing the upper envelope of a set of bats. In the following section, we derive such an algorithm.

3 The Upper Envelope of a Set of Bats

In this section we show how to compute the upper envelope of a set of bats using a sweep line algorithm. This algorithm is essentially a modification of Fortune's algorithm for computing additively-weighted Voronoi diagrams and

Voronoi diagrams of line segments [7]. We say that a point p in the core of some bat is *redundant* if p is contained in some other bat. We say that the input is *redundant* if some bat contains a redundant point and the input is *non-redundant* otherwise.

It is clear that solving the decision problem associated with detour is equivalent to determining whether the input is redundant or non-redundant. This is fortunate, since the upper envelope of n bats can have $\Omega(n^2)$ complexity (see Fig. 2), so any approach that requires computing the upper envelope is doomed to have quadratic running time in the worst case.

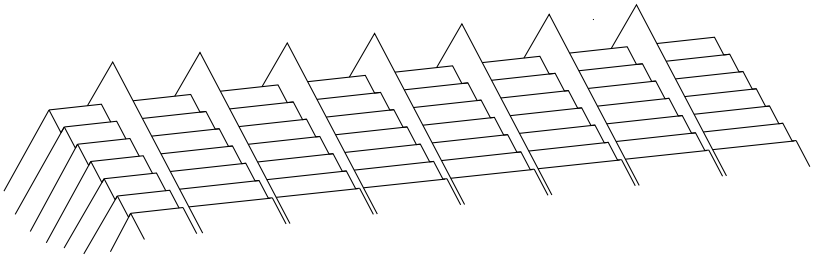

Fig. 2. The upper envelope of n bats can have $\Omega(n^2)$ complexity.

We describe an algorithm that takes as input a set of n bats which are the union of cones that span angles of $2\alpha \leq \pi/2$ and either reports that the input is redundant or correctly computes the upper envelope of the input. The algorithm sweeps a plane P through space and maintains, at all times, the intersection of P with the upper envelope E. The plane P is parallel to the x-axis and forms an angle of $\pi - \alpha$ with the xy plane (see Fig. 3). The reason we sweep with such a plane is that no bat b can contribute to $P \cap E$ until P has swept over some point in the core of b.

To understand the structure of $P \cap E$, it is helpful to note that the boundary of a bat consists of four pieces (see Fig. 4): two conic pieces and two planar pieces. It is easy to verify that the intersection $P \cap E$ is a weakly x-monotone curve consisting of pieces of parabolas and lines. Therefore, its pieces can be stored in a balanced binary tree sorted by x-coordinate. The intersection $P \cap E$ consists of parabolic arcs (where P intersects conic pieces) and straight line segments (where P intersects linear pieces).

As P sweeps through space, $P \cap E$ changes continuously. Therefore, we store $P \cap E$ symbolically so that each arc and segment is represented by its equation as a function of the position of the plane P. The progress of the sweep plane is controlled by a priority queue Q. Initially, Q contains $2n$ events corresponding

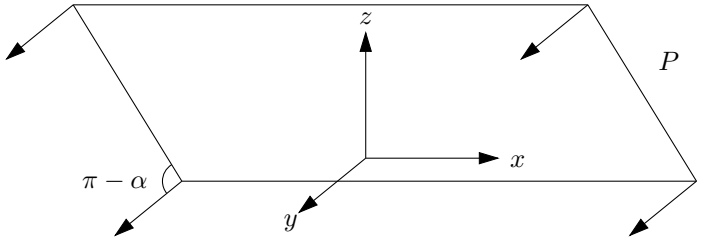

Fig. 3. The sweep plane makes an angle of $\pi - \alpha$ with the xy plane.

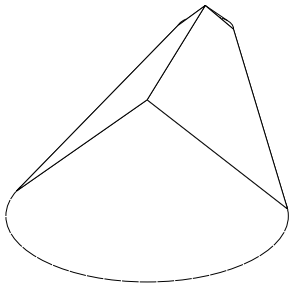

Fig. 4. The boundary of a bat consists of two conic pieces and two planar pieces.

to the times at which the sweep plane passes over each endpoint of the core of a bat.

During the sweep, some arcs or segments of $P \cap E$ may disappear as they become obscured by neighbouring arcs. Since each arc and segment is parameterized by the position of the plane P, it is a constant time operation to determine the time (if any) that an arc will be obscured by its neighbours. In the following discussion, when we insert and delete arcs and segments from $P \cap E$, it is implicit that we recompute the times at which arcs in the neighbourhood of the inserted/deleted arc are obscured and insert these times into Q. For further details refer to Fortune's original paper [7].

During the sweep, we process three types of events:

1. *P sweeps over a point p that is the first endpoint of the core of some bat b.* Refer to Fig. 5.a. In this case, we first check if p is below $P \cap E$. If so, then p is contained in the bat that intersects P directly above p, so we quit and report that the input is redundant. Otherwise, we add four objects to $P \cap E$. These objects are two line segments representing the intersection of P with the two planar pieces of b and two parabolic arcs representing the intersection of P with the cone whose apex is at p.
2. *P sweeps over a point p that is the last endpoint of the core of some bat b.* Refer to Fig. 5.b. In this case, we add two parabolic arcs to $P \cap E$ that correspond to the intersection P with the cone whose apex is at p.

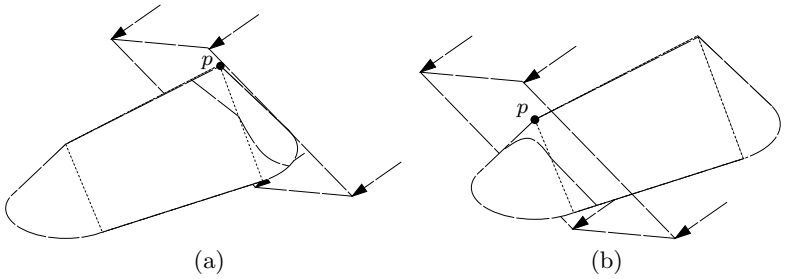

Fig. 5. Handling type 1 and type 2 events.

3. *An arc or segment disappears from* $P \cap E$. In this case, we remove from Q any events associated with the arc or segment. In the case of a segment, we also check if one endpoint of the segment corresponds to a point in the core of a bat. If so, then that point is not part of E so we can quit and report that the input is redundant.

To see that the above algorithm runs in $O(n \log n)$ time, we observe that there are only $O(n)$ type 1 and 2 events. Each of these can be easily implemented in $O(\log n)$ time, and each such event adds $O(1)$ arcs or segments to $P \cap E$. Each type 3 event can also be implemented in $O(\log n)$ time, and deletes one element from $P \cap E$. Therefore, there are only $O(n)$ type 3 events and the entire algorithm runs in $O(n \log n)$ time.

To see that the algorithms is correct for non-redundant inputs we can use arguments which are standard by now [7]. In particular, we can show that there is a direct correspondence between the events processed by the algorithm and changes to the combinatorial structure of $P \cap E$. Suppose therefore that the input is redundant and let p be the first redundant point swept over by P. Either p is an endpoint of a core or it is in the interior of a core. In the former case, it will be handled as a type 1 event while in the latter case it will be handled as a type 3 event.

In either case, all input previously swept over by P is non-redundant, so the intersection $P \cap E$ has been correctly computed. If p is an endpoint of a core it will then be processed as a type 1 event and the algorithm will correctly detect that p is below $P \cap E$ and is therefore redundant. If p is in the interior of a core it will be processed as a type 3 event and the algorithm will correctly detect that the input is redundant. Therefore, either the algorithm correctly computes the upper envelope (if the input is non-redundant) or correctly reports that the input is redundant.

Lemma 1 *There exists an algorithm requiring* $O(n \log n)$ *time and* $O(n)$ *space that tests whether a set of* n *bats is redundant or non-redundant.*

4 Optimization

So far we have given all the tools required for solving the decision problems associated with finding the maximum detour and spanning ratio of a path. More specifically, we have solved the problem: Given a set of segments (possibly points) in 3 space, does there exist a cone c with angular radius $2\arctan(1/d)$, center of rotation parallel to the z-axis and apex on one of the segments such that c intersects another segment. The optimization problem is that of finding the largest value of d for which such a cone exists.

To solve the optimization problem we apply the randomized reduction of Chan [4], which requires only that we (1) be able to solve the decision problem in $f(n) = \Omega(n^\epsilon)$ time, for some constant $\epsilon > 0$, and (2) partition the problem into r subproblems, each of size at most αn, $\alpha < 1$ such that the optimal solution is the maximum of the solutions to the r subproblems. The reduction works by considering the subproblems in random order and recursively solving a subproblem only if its solution is larger than the current maximum (which can be tested by the decision algorithm). The resulting optimization algorithm has running time $O(f(n))$.

We have already shown how to do (1) in $f(n) = O(n \log n)$ time. To do (2), we simply note that we can partition our set of segments in to three sets A, B, and C, each of size $n/3$. The optimal solution is then the maximum of the solutions to $A \cup B$, $B \cup C$ and $A \cup C$. Since there are only $r = 3$ subproblems and each has size $\alpha n = \frac{2}{3}n$ we have satisfied the conditions required to use Chan's optimization technique.

Theorem 1 *The maximum detour and spanning ratio of a planar path with n vertices can be computed in $O(n \log n)$ expected time.*

5 Trees and Cycles

In this section we show how the tools developed for planar paths can be used for solving problems on more complicated types of objects.

5.1 Planar Trees

Let T be a tree embedded in the plane and assume T is rooted at a vertex v such that $T \setminus \{v\}$ has no component with more than $n/2$ vertices. Such a vertex is easily found in linear time. Refer to Fig. 6. We partition the children of v into two sets A and B. Let T_A, respectively T_B, denote the tree induced by v and all vertices having ancestors in A, respectively B. The partition A, B is chosen so that $\frac{1}{4}n \leq \|T_A\|, \|T_B\| \leq \frac{3}{4}n$. Since no descendent of v is the root of a subtree with size more than $\frac{n}{2}$, such a partition can be found with a greedy algorithm.

We lift T into a 3-dimensional tree T' in the following way. Each point $p \in T_A$ is assigned a z-coordinate equal to $\|vp\|_T$. Each point $p \in T_B$ is assigned a z-coordinate equal to $-\|vp\|_T$. This gives us a 3-dimensional tree T' consisting of points T'_A above the xy plane and points T'_B below the xy plane.

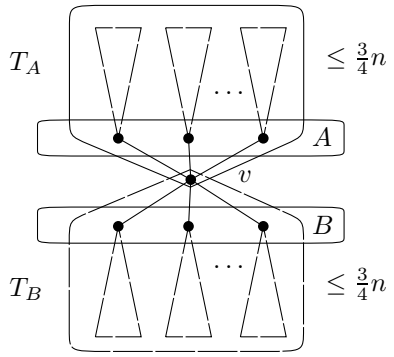

Fig. 6. Partitioning into subtrees T_A and T_B.

This lifting has the property that for any point $p' \in T'_A$ and any point $q' \in T'_B$ the distance between the corresponding points p and q in T is equal to the difference in z-coordinates of p' and q', i.e., $\|pq\|_T = \|p'_z - q'_z\|$. Furthermore, for two points p, q both in T_A or both in T_B, $\|pq\|_T \geq \|p'_z - q'_z\|$. It follows that if we run the maximum detour algorithm of the previous section on the tree T', the algorithm will respond with the correct answer $D(T)$ if there is a pair of points $p \in T_A$ and $q \in T_B$ such that $D(T) = D(T, p, q)$. If this is not the case, then the algorithm may report a value less than $D(T)$, but will never report a larger value.

Therefore, we can compute $D(T)$ using a recursive algorithm. We run the maximum detour algorithm on the tree T', make two recursive calls on T_A and T_B and output the maximum of the three values obtained. To see that this algorithm correctly computes $D(T)$, consider the pair $p, q \in T$ that maximizes $D(T, p, q)$. If $p \in A$ and $q \in B$ (or vice versa), the correct value of $D(T, p, q)$ is found when we run the maximum detour algorithm on T'. Otherwise, $p, q \in T_A$ or $p, q \in T_B$ and is correctly reported by one of the recursive calls.

The running time of the above algorithm is given by the recurrence $T(n) = T(n-k+1) + T(k) + O(n \log n)$, with $\frac{1}{4}n \leq k \leq \frac{3}{4}n$, which solves to $O(n \log^2 n)$.

Theorem 2 *The maximum detour or spanning ratio of a planar tree with n vertices can be computed in $O(n \log^2 n)$ expected time.*

5.2 The Spanning Ratio of a Planar Cycle

To obtain an algorithm for computing the spanning ratio of a planar cycle, we study the following decision problem in $\mathbb{R}^3$: Given a set S of n points in $\mathbb{R}^3$, do there exist two points $p, q \in S$ such that $(p_z - q_z)/\|p_{xy}q_{xy}\| > d$ and $p_z - q_z < 1/2$? This problem is almost identical to our previous problem on cones except that now, instead of being infinite, the cones have height $1/2$.

To reduce the problem of computing the spanning ratio of a polygonal planar cycle C to the above problem we first normalize the cycle C so that it has length

1. We then remove an arbitrary edge of C so that it becomes a path and lift the vertices of this path into $\mathbb{R}^3$ as described in Section 2. This gives us a set S_1 of n points in $\mathbb{R}^3$. Next, we make a copy S_2 of S_1 and translate S_2 downwards (in the $-z$-direction) by a distance of 1. The union S' of S_1 and S_2 consists of $2n-1$ points. Then it is not hard to verify that the spanning ratio of C is larger than d if and only if there exists $p, q \in S'$ or such that $(p_z - q_z)/\|p_{xy}q_{xy}\| > d$ and $p_z - q_z \leq 1/2$.

So far we have reduced the problems of computing the spanning ratio of a planar cycle to a problem on finite cones. Unfortunately, this problem is very different from our previous problem that involved infinite cones, and can not be solved by computing the upper envelope of the cones. However, suppose we have a dynamic data structure that supports insertion and deletion of (infinite) cones and signals when the set of cones currently contained in the data structure is redundant.

Then we can solve our problem on finite cones by sweeping with a plane P that is parallel to the xy plane and maintaining the data structure so that it contains only the cones with apexes on points of S that are below P but at distance at most $1/2$ from P. If at any time the data structure reports that the input is redundant then we know that there are two points $p, q \in S$ such that $(p_z - q_z)/\|p_{xy}q_{xy}\| > d$ and $p_z - q_z \leq 1/2$. On the other hand, if two such points p and q exist, then they will both be in the data structure at some point in time and the data structure will report that the input is redundant. Since only $O(n)$ points are inserted and deleted in the data structure, this algorithm will run in $O(n \log n + nU(n))$ time, where $U(n)$ is the time it takes to perform an update operation on the data structure.

All that remains is to develop a data structure that supports insertion and deletion of cones and signals if, at any time, the set of cones currently contained in the data structure is redundant. In general, this can be treated as a decomposable search problem and a data structure with $O(n^{\frac{3}{2}} \log n)$ update time can be obtained using the technique of Bentley and Saxe [2]. However, in our application insertions and deletions are done in a FIFO manner, so that the ith point inserted will be the ith point deleted. We will use this property to obtain a data structure that supports updates in $O(\log^2 n)$ amortized time.

The data structure maintains a partition of the cones into $O(\log n)$ sets. These sets are denoted by $I_0, \ldots, (I_k = D_k), \ldots, D_0$ where I_i, respectively D_i, either contains exactly 2^i cones or is empty. At all times, the data structure maintains two invariants: (1) The set $I_k = D_k$ is non-empty, and (2) all the cones in I_{i+1} will be deleted before any of the cones in I_i, and all the cones in D_i will be deleted before any of the cones in D_{i+1}, for all $0 \leq i \leq k$.

When inserting a cone c into the data structure, we check if the apex of c (and hence all of c) is contained in any cone of the data structure. If it is, then the input is redundant and we can quit. We can perform this test efficiently by maintaining, for each I_i (respectively, D_i), the upper envelope of the cones in I_i stored in a point location structure that can test, in $O(\log n)$ time if a point $p \in \mathbb{R}^3$ is above or below the envelope.

After performing this test, and assuming it is negative, we insert the cone c into the data structure by finding the smallest value of i such that I_i is empty. If no such value of i exists, we increase the value of k by 1 so that I_k is empty. We group together the cone c and all the cones contained in $I_0, \ldots, I_{i-1}$, place these in I_i and build the upper envelope for I_i. At the same time, we make the sets $I_0, \ldots, I_{i-1}$ empty. It is clear that this maintains invariants 1 and 2.

To delete a cone from the data structure we find the smallest value of i such that D_i is non-empty. Because of invariants 1 and 2, D_i must exist and contain the cone c that is being deleted. We then partition $D_i \setminus \{c\}$ into $D_{i-1}, \ldots, D_0$ in such a way that invariant 2 is maintained, build the data structures for $D_{i-1}, \ldots, D_0$ and make D_i empty. To maintain invariant 1 we check if $i = k$. If $i = k$ and I_{k-1} is empty then we decrease the value of k by 1. Otherwise if $i = k$ and I_{k-1} is not empty then we merge I_{k-1} and D_{k-1}, put the result into $I_k = D_k$ and make I_{k-1} and D_{k-1} empty.

A simple amortized analysis of this data structure, which we don't include because of space constraints, shows that the amortized cost of all operations is $O(\log^2 n)$. (Give each cone $2\log^2 n$ credits when it is inserted and make it pay $\log n$ credits every time it moves.) Combining this with the machinery of Chan's optimization technique, we have just proven Theorem 3.

Theorem 3 *The spanning ratio of a planar cycle with n vertices can be computed in $O(n \log^2 n)$ time.*

5.3 The Maximum Detour of a Planar Cycle

To compute the maximum detour of a planar cycle we make use of the following lemma, which is a generalization of a lemma by Ebbers-Baumann *et al* [5].

Lemma 2 *Let C be a planar cycle of length 1. Then there exist two points $s, t \in C$ such that $D(C, s, t)$ is maximal and*

1. *s is a vertex of C or*
2. *(2) $\|st\|_C = 1/2$.*

Proof. The proof is omitted due to space constraints.

Given a planar cycle C of length 1, we can find the maximum detour among all points $s, t \in C$ such that $\|st\|_C = 1/2$ in linear time by starting with any two points s and t and sweeping them around C while maintaining the invariant that $\|st\|_C = 1/2$. Once we have done this, Lemma 2 implies that we can limit our search to pairs $s, t \in C$ such that s is a vertex of C.

Given this result, we can proceed in the same manner as we did when computing the spanning ratio of a planar cycle, except that now our dynamic data structure maintains a set of bats and the upper and lower envelopes are constructed using the algorithm of Section 3. Although the basic approach is essentially the same, two technical difficulties occur. Due to space constraints, we only sketch their solution.

The first difficulty is that bats in our data structure change continuously as the sweep plane sweeps over the cores of bats. However, Lemma 2 allows us to discretize this change by inserting, for each vertex v of C, a *Steiner vertex* v' whose distance from v along C is exactly $1/2$. The second difficulty is that the plane sweep will only find the pair s, t with maximum detour if the shortest path from s to t is counterclockise around C. To handle this problem we perform two plane sweeps, one in the $+z$ direction and one in the $-z$ direction.

As before, if at any time during the maintenance of the data structure we find some subset of bats to be redundant then we can quit. When inserting a bat b into the data structure, Lemma 2 ensures that performing point-location queries on the endpoints of the core of b is enough to ensure the correctness of the algorithm. Leaving further details to the full version, we obtain:

Theorem 4 *The maximum detour of a planar cycle C with n vertices can be computed in $O(n \log^2 n)$ time.*

References

1. P. K. Agarwal, R. Klein, C. Knauer, and M. Sharir. Computing the detour of polygonal curves. Unpublished Manuscript, November 2001.
2. J. L. Bentley and J. B. Saxe. Decomposable searching problems. I. Static-to-dynamic transformation. *Journal of Algorithms*, 1(4):301–358, 1980.
3. P. Bose and P. Morin. Competitive online routing in geometric graphs. In *Proceedings of the VIII International Colloquium on Structural Information and Communication Complexity (SIROCCO 2001)*, 2001.
4. T. M. Chan. Geometric applications of a randomized optimization technique. *Discrete & Computational Geometry*, 22(4):547–567, 1999.
5. A. Ebbers-Baumann, R. Klein, E. Langetepe, and A. Lingas. A fast algorithm for approximating the detour of a polygonal chain. In *Proceedings of the 9th Annual European Symposium on Algorithms (ESA 2001)*, pages 321–332, 2001.
6. D. Eppstein. Spanning trees and spanners. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 425–461. Elsevier, 1999.
7. S. J. Fortune. A sweepline algorithm for Voronoi diagrams. *Algorithmica*, 2:153–174, 1987.
8. C. Icking and R. Klein. Searching for the kernel of a polygon: A competitive strategy. In *Proceedings of the 11th Annual Symposium on Computational Geometry*, pages 258–266, 1995.
9. W. A. Johnson and R. F. Mehl. Reaction kinetics in processes of nucleation and growth. *Transactions of the Americal Institute of Mining and Metallurgy*, 135:416–458, 1939.
10. G. Narasimhan and M. Smid. Approximating the stretch factor of Euclidean graphs. *SIAM Journal on Computing*, 30(3):978–989, 2001.